I Own, I Provide, I Decide: Generalized User-Centric Access Control Framework for Web Applications

Kapil Singh, Ikpeme Erete and Wenke Lee
School of Computer Science, Georgia Institute of Technology
{ksingh, ikpeme, wenke}@cc.gatech.edu

ABSTRACT

With the rapid growth of Web 2.0 technologies, users are contributing more and more content on the Internet, in the form of user profiles, blogs, reviews, etc. With this increased sharing comes a pressing need for access control policies and mechanisms to protect the users' privacy. Access control has remained largely centralized and under the control of the web applications hosted on their servers. Moreover, most web applications either provide no or very primitive and limited access control. We argue that the owner of any piece of data on the web should be able to decide how to control access to this data. This argument should hold not only for the web applications contributing data, but also for the contributing users. In other words, users should be able to choose their own access control models to control the sharing of their data independent of the underlying applications of their data.

In this work, we present a novel framework, called `xAccess`, for providing generic access control that empowers users to control how they want their data to be accessed. Such a control could be in the form of user-defined access categories, or in the form of new access control models built on top of our framework. On one hand, `xAccess` enables individual users to use a single unified access control across multiple web applications; and on the other hand, it allows an application to support different access control models deployed by its users with a single model abstraction. We demonstrate the viability of our design by means of a platform prototype. The usability of the platform is further evaluated by developing sample applications using the xAccess APIs. Our results show that our model incurs minimum overhead in enforcing the generic access control and requires negligible changes to the application code for deployment.

1. INTRODUCTION

With the advent of Web 2.0 technologies, web application development has become much more distributed with a growing number of users acting as developers and sources of online contents. In particular, many users are contributing more and more contents, by providing their personal information on social networks or by adding information in the form of blogs, reviews, etc.

While the trend is towards more user-contributed data, the mechanisms to define the access control policies on user-contributed data are still under the control of the web applications. Consider the example of social networks such as Facebook. Users contribute data in the form of their profile information, by loading pictures, or by posting messages in each other's profiles. The mechanism to control access to the users' data is determined by the social networking web site; in most cases, it is limited to a small number of pre-defined access categories such as private, public or to providing access only to the users' friends. As a result, the users are forced to use alternate means to protect their privacy, for example, by maintaining multiple blogs [16].

Even if a web site or application wants to change its access control mechanism to satisfy the needs of its users, there are major obstacles. First, the diversity in the user population and the variety of the data contributed by each user means that developing a mechanism that caters to the need of every user might not be feasible. Second, even though the privacy expectations that users desire are easy to state, there is still a large gap between users' mental models and the policy languages of the current access control systems provided by the applications [4].

We argue that the users are the ones best suited to decide the semantics and importance of their own data. Thus, we need to provide users the freedom and utility to define the access control policies specific to their data, and enable web applications to enforce these potentially diverse policies.

In this paper, we propose a generalized framework for providing access control in web applications that provides users with a wide range of access control options to satisfy their own individual privacy requirements and is independent of any application. In our design, the users have the flexibility to define their own policies to control the privacy of the data contributed by them. A simple example of such user-specific policies for a blogging application is that of a blogger defining different access to his blog for personal friends and office colleagues. Another user of the same application may only desire public or private access for his blog entries. Our framework provides an abstract base model to which the user's access control policies can be mapped: this single, unified abstraction allows the modeling of a wide range of access control policies specified by different users of an application.

.

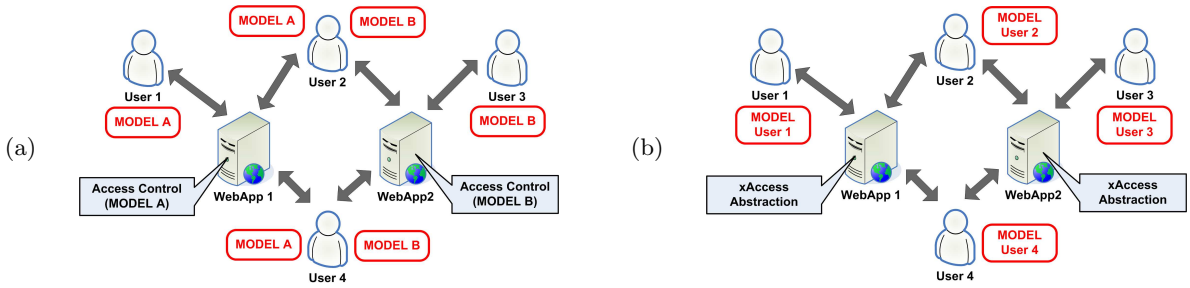

Figure 1: Application architecture for: (a) current frameworks. (b) xAccess framework.

We demonstrate the viability of our design by implementing a prototype system, called xAccess. xAccess has two components, one that runs on the client side as an extension to the user's browser and another component that is hosted on the server of the web application. A user desiring access control for his data defines his policies using an interface provided by the xAccess extension. The extension translates these user policies to categories in our base access control model. Our model is based on the Role Based Access Control (RBAC) scheme, so these categories translate to specific roles in the model. xAccess also provides an interface to apply the user-defined policies to any granularity of data desired by the owner, for example, to protect individual blog entries, particular personal information, or specific photos. Furthermore, xAccess also allows specific words or phrases within a blog to be tagged with user-defined categories, thereby allowing data owners to control access to specific information, such as someone's name in a blog entry. Only a user who wants to control access to his data is required to install the xAccess' browser extension; the extension is not needed by any other user who is seeking access to the owner's data. For the rest of the paper, we use the term *owner* to denote an individual providing data and *seeker* to denote a person who wants access to the owner's data.

The server-side component of xAccess uses the access categories of the base model to determine whether a particular access should be granted. Our model not only controls access to the read operation, but also supports access control to other operations like write or download. In a blogging application, the server-side component filters a blog considering the categories attached to various parts of the blog. By default, a reader is only presented with the public entries of an owner's blog. Further access is granted only after the owner's policy specifies a category label for the reader and the access is limited to that category. In a wikipedia application, the owner can similarly restrict write operation to his wiki entry by attaching appropriate write permission to the entry.

One of the strengths of xAccess' abstract model is that it allows other access control models to be incorporated into our framework. This is a desirable feature, because we expect other current and future access control models to facilitate development of more diverse user policies closer to an owner's mental model. Our base model is generic and can simulate a wide range of such models (Section 3.2). For example, in the blogging application, new models such as CBAC [14] can support policies like "only people mentioned in the blog should see the blog". Our framework supports such models, thereby supporting more diverse user policies, without requiring any change to our base model and without any modifications to the web application.

1.1 Limitations of Current Designs

The current access control design of web applications enforces an application-specific single security policy for every user of that application. However, there are different degrees of intimacy between an individual and any other user desiring access to his data, but the current designs are too coarse to capture these distinctions. In most current access control systems, the owner of data has no say in defining the granularity of access allowed.

These coarse-grained policies might result in undesired access or undesired restrictions. For example, users on Flickr only have the option to assign their office colleagues to the category of family, friends, or public. Assigning them public access would prevent them from viewing some of the "professional" pictures posted by the user. On the other hand, giving them friends access would allow them to see any pictures accessible to friends, which might not be desirable for some users. On the flip side, some users might not want to share some of the professional pictures with a few or all of their friends or family due to confidentiality constraints. The web application is ill-equipped to decide the access granularity required by a particular owner; only the owner is familiar with his own particular situation and relationships to provide the right level of access to other users in the system. Most web applications currently employ limited number of access control categories with little or no flexibility in adding new user-specific categories.

The web application is also restricted in terms of the access control mechanism it uses. In the current setting, most applications provide a single access control model for all users of the application (Figure 1(a)). For example, Facebook uses access control lists to evaluate access to a user's profile, photos, messages, etc. This mechanism is common for all the users, independent of individual requirements. Additionally, the mechanism is more-or-less fixed and there is no incentive for web applications to change the model unless there is a huge user base vouching for that change. As a result, new innovative mechanisms such as CBAC [5] might take years before being adapted by the web applications. Even if some applications are more proactive to the change, it does not hold true for most web applications.

1.2 Our Contributions

Our framework design allows an owner to keep a single access control model that can be utilized for one or multiple web applications (Figure 1(b)). However, the owner also has

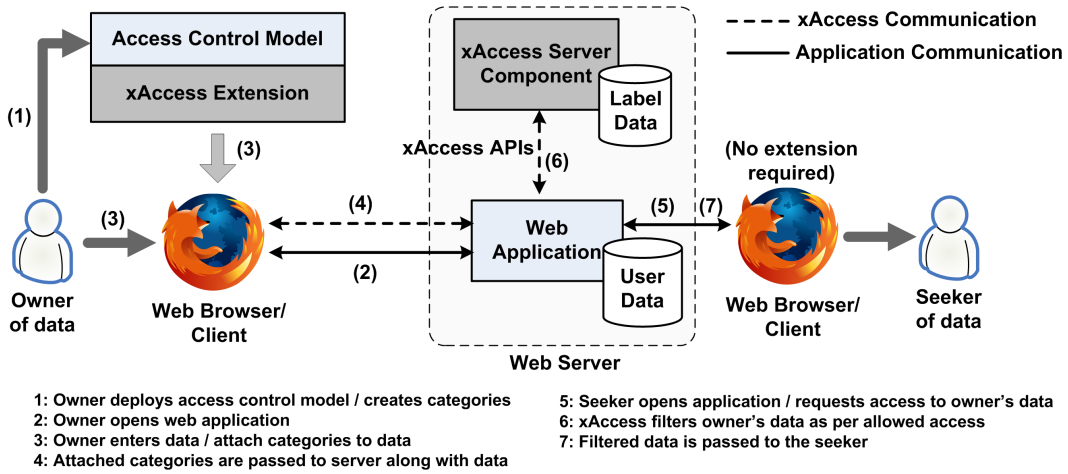

Figure 2: `xAccess` Architecture and workflow scenarios.

the option to use different models for different applications. For example, the framework empowers the owner to deploy access control mechanisms like MAC [24] or CBAC [14] allowing him flexibility in writing his own access control policies. From any web application's prospective including the ones that currently do not employ access control, one-time installation of our framework enables the application to support different access control models for different owners.

Our work makes the following contributions:

- We propose a novel design of an access control framework for supporting diverse user-defined access control policies that empowers the users to choose their own models and their own access granularity. We also show that our model is *generic* to allow simulation of a number of popular access control models on top of our framework, and provides enormous *flexibility* to the users in making access control decisions about the data owned by them.
- We develop a proof-of-concept system, called xAccess (extended Access), that provides a set of APIs that can be used to integrate generalized access control capability into web applications.
- We demonstrate the viability of our framework by developing a sample blogging application as our base example and subsequently integrating access control into the application using the xAccess APIs. We also show real-world deployment potential of our framework by integrating xAccess into a popular open-source wikipedia application. Our sample web applications are available online [1, 3].

Paper Organization. The rest of the paper is organized as follows. We present an overview of the xAccess framework in Section 2. We present the base model for access control in xAccess in Section 3. Section 4 presents the implementation details and evaluation of xAccess. We discuss the advantages and limitations of xAccess in Section 5. Related work is presented in Section 6, followed by conclusions in Section 7.

2. THE `xAccess` FRAMEWORK

xAccess is a framework for enabling web applications to capture and model data owners' privacy policies and to enforce such policies via access control on data seekers. xAccess is designed to be general and adaptive so that it can be used for a wide variety of web application scenarios and more importantly for different access control models implied by owners' diverse policies. More specifically, xAccess only requires one-time installation at the server side; after this initial installation, no change is required at the server and owners are free to change their policies via a client side component at any time. Furthermore, xAccess provides a generalized access control model, based on the Role Based Access Control (RBAC) model [12, 25], to represent policies specified either in the form of user-defined access classifications (such as private, friends, family, etc.) or in the form of other access control models (such as MAC [24], DAC [23], CBAC [14], etc).

Figure 2 shows a high-level design of the xAccess framework. There are two parts of the xAccess platform, one that is hosted on the server-side and the other that runs on the client-side as an extension to the data owner's web browser.

On the client-side, the xAccess extension provides a layer of abstraction that enables its RBAC-based generalized access control model (which we call *base model*) to directly represent policies specified in terms of user-defined access categories as well as convert policies specified using other access control models. As we will show in Section 3.2, widely used access control models, including Discretionary Access Control (DAC), Mandatory Access Control (MAC) and Role Based Access Control (RBAC), can be constructed on top of the base model in xAccess. These models are simulated in xAccess by setting various parameters of the base model using the APIs provided by xAccess.

More generally, as shown in Figure 2, any owner-specific access control model, e.g., CBAC, is deployed as a layer above xAccess and utilizes its APIs. When deployed, such a model is coupled with xAccess in a web browser extension. If no such access control model is already designated by a data owner, which is the more common case, xAccess' browser extension presents a user interface that allows him to define access categories directly into the base model. Additionally, the browser extension also provides an interface for a data owner to assign categories to any information he puts in a web application. This interface forms part of

the browser's chrome and does not modify the application's code. It supports access control for both structured information (such as well-defined fields in a user's personal profile) and unstructured data (such as blogs). The xAccess extension only needs to be installed at the owner's web browser, and is not required for any other user only seeking access to the owner's data.

The mapping of the entities (both subjects and objects) with their corresponding categories is passed to the application's server. The server-side component of xAccess receives and stores these mappings. The xAccess component is a separate module on the server side; it is only invoked by the web application when required to filter content that is access controlled (Figure 2). The amount of modifications required to integrate xAccess into an application is negligible (3–5 lines of API calls).

The enforcement of access control is done by the server side component of xAccess, which serves all users of the application and in effect realizes their corresponding access control models. Since the translation of user-specific access control models to xAccess' base model is done at the client side and only the access categories corresponding to the base model are presented to the server, this greatly simplifies the access control enforcement at the server side. xAccess' server-side component uses a simple matching algorithm to decide if a requesting subject (seeker) is allowed to access an owner's data object. The algorithm uses categories of the subject and the object in making such a decision. We present details of this algorithm when we introduce our model in Section 3.1.

In our current design, the server side of xAccess is invoked as a set of API calls made by the web application. An alternate design would be to deploy xAccess as a proxy that filters data before passing it to the user's browser. The difference from our current design is subtle and a detailed comparison is not the focus of our work.

3. USER-CENTRIC ACCESS CONTROL

In this section, we present the generic access control model that underlines our xAccess framework. We call this the base model. It has the following design goals:

- **Generalization.** Since one of the goals of the xAccess framework is to allow data owners to specify their access policies, which can be very diverse, the base model should be able to accommodate a wide range of access control models. That is, it should allow policies defined using different access control models to be expressed or simulated in the base model. It should also model policies expressed in terms of access categories, as commonly used in web application scenarios.
- **Minimum modification requirements.** Once a web application deploys the xAccess framework, it should require no further changes to the application even if a data owner changes his access model at any later time.
- **Backward compatibility.** Even after deploying the xAccess framework, an application should still support users with no xAccess component installed on the client-side. In other words, the fall-back mechanism of xAccess should be the same (default) behavior of the application when no xAccess is deployed.

In order to achieve these goals, we designed xAccess to be policy neutral. The base model in xAccess provides an abstraction of the essential elements of any access control policy, which would at the minimum include the access categories or role hierarchies, and the constraints and administration of user-role and role-permission assignments. We next describe the base model, and show how it enables diverse user policies to be modeled and enforced in the xAccess framework.

3.1 The Base Model

We constructed our model by customizing the Role Based Access Control (RBAC) model for our *user-centric* paradigm. The central idea of RBAC is that permissions are associated with roles and users are made members of appropriate roles thereby acquiring the roles' permissions. Roles are created for the various job functions in an organization, and users are assigned roles according to their responsibilities and qualifications. The roles are assigned by the system administrator of the organization. RBAC allows for the specification and enforcement of a variety of protection policies, which can be tailored on an enterprise-by-enterprise basis. The RBAC framework provides administrators with the capability to regulate who can perform what actions, when, from where and in what order.

The goal of the xAccess framework is to provide protection to user data in accordance with the access control defined by the data owner. In this regard, user's contributed data (i.e., user profile, blogs, photos, etc.) for a particular web application represents a habitat that corresponds to an organization in the RBAC model. In our model, we associate the administrator privileges for the access control over any data items to the owner of those items. The data owner defines the roles for his "system" because he is the one who knows the "responsibilities and qualifications" of various individuals, acting as users of the application, from his personal connections to people. For example, a data owner knows who his family members are in real life and how much each person can be trusted with his data. In other words, a role in our model signifies a real-life relationship of the data owner. This relationship could be in the form of family, friends, business colleagues, public or any others role specific to the data owner. Such roles might vary from user to user. There is many-to-many mapping from roles to users; this also shows similarity to the the real world, where an individual might have multiple friends or family members, or a friend could also be part of the family thereby assuming both roles.

In RBAC, a role signifies a set of operations that can be performed by that role. In our model, these operations can take limited forms depending on the application. In most cases, the operation would be limited to data read, allowing a data owner to control the confidentiality of his data. Some typical applications include social networks and blogs. In other cases, the data owner might want to assign write permissions to the data contributed by him. Wikipedia is one application that falls into this category. Providing access control for other operations such as direct download, remote execution, etc. is also feasible in our system, if such operations are supported by the application.

The xAccess framework tracks and enforces access control using a labeling system defined based on the existing RBAC models [12,25]. All system abstractions are layered on top of two types of entities – subjects and objects. Subjects represent the users requesting data access and objects represents

the data entities that are being requested. Both subjects and objects correspond to roles in the xAccess framework.

Subjects are associated with roles when the request for access is granted by the web application on behalf of the data owner. The objects are assigned roles when data is entered into the application by the data owner, e.g., by uploading new photos, writing new blog entry, adding new profile information, etc. A data owner can update roles of both subjects and objects anytime at his own discretion.

3.1.1 Formal Description

For any user u of a particular application A, an xAccess system consists of a set of subjects $S_u \subseteq \mathbb{S}_\mathbb{A}$ and a set of objects $O_u \subseteq \mathbb{O}_\mathbb{A}$, where $\mathbb{S}_\mathbb{A}$ is a set of all registered users of the application and $\mathbb{O}_\mathbb{A}$ represents a set of objects allowed for the application. $\mathbb{O}_\mathbb{A}$ includes both structured (e.g., user profile) and unstructured (e.g., blog entry) data items specific to the application A.

Additionally, a data owner u's generic access control model allows access to a set of roles $R_u \subseteq \mathbb{R}$ and the corresponding permissions $P(r)$ associated with each role $r \in R_u$. Here, $\mathbb{R}$ represents a complete set of possible roles created by a data owner according to his real-life relationships. Let us assume that the application A permits operation set $OP_A(T)$ for any data seeker (who logs into the application) requiring access to a component T of another user's data. T represents a unit of data with unique allowed operations and depends on the internal logic of a particular application. For example, user profile and blog entry represent two such example components. An application, say A, may allow a user to edit his own profile, but only allows reading of other user's profile, thereby implying $OP_A(profile) = \{read\}$. However, it may allow both read and write operations on the blog ($OP_A(blog) = \{read, write\}$). $COMP(o)$ represents the application component that contains object o.

Therefore, for an application A and any role $r \in R_u$, the operations that any seeker can perform on data owner u's data object o is a set $P_{effective}(r, o) = P(r) \bigcap OP_A(COMP(o))$. Both subjects and objects are associated with set of one or multiple roles, defined by the function $role : S_u \rightarrow R_u$ for subjects and $role : O_u \rightarrow R_u$ for objects of data owner u.

Basic Access Rule A subject $s \in S_u$ can perform an operation op on an object $o \in O_u$ if and only if $\forall r \in role(s), \exists r' \in role(o)$ such that $r = r'$ and $op \in P_{effective}(r, o)$.

3.1.2 Role Hierarchy

Roles can have overlapping responsibilities and privileges, that is, users belonging to different roles may need to perform common operations on some objects. For example, a best friend should be able to read the data accessible to a friend's role. Furthermore, often there are a number of data items that can be accessed by all roles in their system. As such, it would prove inefficient and administratively cumbersome to repeatedly add access to these objects for each role that gets created. To overcome this limitation, our model includes specification of role hierarchies. A role hierarchy defines roles that have unique attributes and that may "contain" other roles. That is, one role may implicitly include the operations, constraints, and objects that are associated with another role. Role hierarchies are a natural way of organizing roles to reflect authority and responsibility, and competency.

An example of a role hierarchy is shown in Figure 3. In this example, the role `Private` "contains" the roles of `Best Friend` and `Family`. This means that members of the role `Private` are implicitly associated with the operations, constraints, and objects of the roles `Best Friend` and `Family` without the administrator having to explicitly list their attributes for the `Private` role. The most powerful roles are represented at the top of the diagram with the less powerful roles being represented at the bottom. That is, the roles on the top of the diagram contain the greatest number of operations, constraints, and objects. As shown in Figure 3, not all roles have to be related. The roles `Friend` and `Family` are not hierarchically related but they can contain some or all of the same roles.

For a data owner u, role hierarchy $RH_u \subseteq R_u \times R_u$ is as a partial order relation on R_u, written as $\succeq$. That is, if $r_1 \succeq r_2$ then role r_1 is higher than (or contains) role r_2. We define our access control rule in presence of the role hierarchy as follows.

Hierarchy Access Rule A subject $s \in S_u$ can perform an operation op on an object $o \in O_u$ if and only if $\forall r \in role(s)$, $\exists r' \in role(o)$ such that $r \succeq r'$ and $op \in P_{effective}(r, o)$.

Considering the example hierarchy in Figure 3, let us assume that a subject u_1 with $role(u_1) = \{Family, Best\ Friend\}$ wants to access (read) a photo object o_1 with $role(o_1) = \{Friend\}$. The application allows only *read* permission to o_1. Since the role of the subject u_1 (*Best Friend*) is higher than the object o_1 (*Friend*) and $P_{effective}(Best\ Friend, o_1) = \{read\}$, u_1 can successfully access the photo o_1. As another example, let us consider another user u_2 with $role(u_2) = \{Friend\}$ who wants to write to a blog entry o_2 with $role(o_2) = \{Public\}$. In this case, the application allows both *read* and *write* permissions to o_2. Even though the role of u_2 (*Friend*) is higher than o_2 (*Public*), u_2 is still not allowed to write to o_2 since $P_{effective}(Friend, o_2) = P(Friend) \bigcap OP_A(COMP(o_2)) = \{read\} \bigcap \{read, write\} = \{read\}$.

xAccess allows new constraints to be added into the model at any stage. Constraints are set of rules that are mandated over any role assignments and hierarchy definitions. They define a broad scope of what is acceptable in the xAccess model. For example, a user may want to enforce a rule that none of his office colleagues can be on his family list. This ensures that information about his family activities or family pictures are kept hidden from his office colleagues. Accordingly, a constrain can be added using the xAccess APIs that enforces that no user can be assigned the twin roles of `Family` and `Office Colleague` at the same time.

Note that xAccess enforces no restriction on what roles can be defined. It only provides a framework that can be utilized to define roles and role hierarchies that can effectively simulate the access control model underlying the user-defined access control policies. Section 3.2 will discuss how our base model can simulate the commonly used access control models, thereby enabling a generic access control system for diverse user and application needs.

The server-side framework of xAccess stores the roles for both subjects and objects associated with each user, the hierarchy of roles, as well as the corresponding allowed operations such as read or write for each role. It also enforces the access control rules on behalf of the data owner. Figure 4 shows the algorithm used in the xAccess framework to evaluate whether or not to grant access to the seeker for

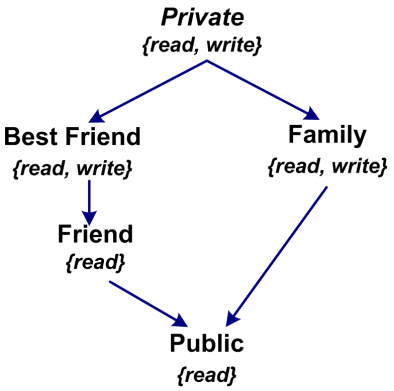

Figure 3: Example role hierarchy in xAccess. Text in *italics* represents corresponding permissions.

Algorithm 1 Access Control Algorithm of xAccess.

```
INPUT S: subject, O: object, REQ_OP: operation requested
if application supports REQ_OP on O then
  R_s ← getRoles(S)
  R_o ← getRoles(O)
  for all r ∈ R_s do
    for all r' ∈ R_o do
      if r ⪰ r' and REQ_OP ∈ getOperations(r) then
        GRANT S with REQ_OP access to O
        return
      end if
    end for
  end for
end if
DENY S with REQ_OP access to O
```

Figure 4: Pseudo code of the algorithm to check if the Subject S can perform operation REQ_OP on Object O. REQ_OP can be in the form of read or write on any other application-specific operation. *getRoles(x)* returns the roles corresponding to the entity x and *getOperations(y)* returns the permissions for the role y.

the requested data item or resource. The algorithm covers *Hierarchy Access Rule* with the condition $r \succeq r'$. Replacing this condition with $r = r'$ will limit the algorithm to support only the *Basic Hierarchy Rule*. The simplicity of this algorithm demonstrates the value of our design where a single algorithm is able to effectively cover a wide range of access control models used by different data owners.

3.2 Expressiveness of the Base Model

One of the requirements of the xAccess framework is that it should allow users to choose their own access control models. In order to fulfill this requirement, the xAccess base model should be generic enough to capture the functionality of the chosen access control model. In other words, the abstractions in the base model should be expressive enough to enforce a wide range of access control models.

For most current web pages, access control is enforced by just customizing the access levels or categories and by providing partial order to such categories. For example, in Facebook such categories are friends, friends of friends, private, etc. These simple policies can be directly modeled by the base model in xAccess. However, we anticipate that future web applications may require more elaborate access control policies.

In this section, we discuss how our RBAC-based model can simulate the other access control models. In particular, we consider few traditional and most commonly used access control models–Discretionary Access Control (DAC), Mandatory Access Control (MAC) and Lattice Based Access Control (LBAC) [10,24]–and their variants, along with some newer models like Content Based Access Control (CBAC) [14].

3.2.1 DAC

Discretionary Access Control or DAC is a means of restricting access to objects based on the identity of subjects and/or groups to which they belong. The controls are discretionary in the sense that a subject with a certain access permission is capable of passing that permission on to any other subject. Sandhu et al. demonstrated that several variations of DAC can be simulated via the RBAC model. The basic idea behind the DAC to RBAC construction is to simulate the owner-centric policies using roles that are associated with each object [23]. In our framework, the owners have discretion to transfer certain controls (read, write or execute) to other users. In our default setup, transfer of controls is allowed only if the original owner provides ownership permission to other users. Depending on his own requirements, the owner can create additional permissions to restrict transfer to only specific controls.

3.2.2 MAC / LBAC

Mandatory Access Control (MAC) refers to a system of access control that assigns security labels or classifications to objects and allows access only to subjects with distinct levels of authorization or clearance. In contrast to DAC, where a subject can pass permissions to access an object to other subjects, no such propagation is allowed in the MAC model. These controls are enforced by the security administrator of the base system.

Lattice Based Access Control or LBAC is a special type of MAC where a lattice is used to define the levels of security that an object may have and that a subject may have access to. A subject is only allowed to access an object if the security level of the subject is greater than or equal to that of the object. For example, in the xAccess system, a user can give different levels of access to top friends, friends or family. Based on how the lattice is constructed, LBAC can be used for confidentiality, integrity, or both.

Osborn et al. have demonstrated that the LBAC model can be simulated using RBAC [21]. Since our base model is a customized form of RBAC, any variations of the MAC model can be transformed to our base model. In our model, an owner acts as the administrator for his own data representing the system administrator in the MAC model.

3.2.3 CBAC

CBAC is a type of access control in which access to an object is partially or entirely based on the content of the

objects in the system [5, 22]. CBAC allows users to specify a single, intuitive access control policy based on object features and then automatically applies that policy to new objects as they are created. CBAC utilizes techniques from natural language processing [22], image processing [5] and machine learning to perform this task. Essentially, it provides the necessary bridge between a user's intuitive access control policy, such as "Parents should not see my party pictures", and the policy enforcement.

CBAC is currently an open research project, with efforts to improve its practical acceptability to a variety of applications [14]. The advantage of CBAC is that it reduces the burden of managing the access control policies for the users – the users just need to provide high-level policies and CBAC controls the policy enforcement with no further intervention required from the user.

The advantage of our xAccess framework is that it allows CBAC to be integrated into a web application easily using our abstractions. It would also enable easy integration of CBAC into the application at any time in the future, i.e., whenever users think that it is viable enough for their purposes. Note that xAccess places no guarantee or control over the correctness of CBAC; the users are empowered to decide whether or not to use the CBAC systems. An alternative to our approach in the current systems is that the applications themselves switch to the CBAC model. However, applications have to cater to the need of all its users to consider a new technology and hence might take much longer to switch to the new CBAC model. Furthermore, switching to a new model again limits all users of the application to that model.

To deploy CBAC on top of our xAccess framework, the implementation of CBAC need to convert high-level user policies in their model to lower level roles, subjects and objects in the xAccess model. For example, let us consider a simple CBAC policy "Only my friends can see my party pictures". First, the CBAC implementation uses image processing to determine which pictures are from "parties" and tag all such pictures with a "friend" role. The user could have approved a list of individuals to be added in the friend role. More advanced implementations of CBAC can analyze the user's data to infer the friend's list of the user based on who talks with whom.

Since the CBAC policies can be effectively implemented using RBAC, we can infer that the CBAC model can be simulated using the xAccess system.

3.3 Access Control Lifecycle in xAccess

We can now summarize the process by which any access control model defined by a data owner by means of the client (browser) component of the xAccess framework is mapped to the enforcement of access control at the server.

1. In the absence of any access control model specified by the user, xAccess uses its own base model by default. The xAccess' browser extension presents a user interface to the data owner for defining his access preferences directly into our model. This interface allows the data owner to create roles and the corresponding role hierarchy. Alternatively, the data owner may choose to deploy his own access control model over xAccess (Figure 2). Such models implement their access control logic by defining roles and roles hierarchies using the APIs provided by xAccess' browser extension. Orthogonally, the implementation of these models can provide their own interface for the owner to define the model-specific policies. We believe that user-friendly interfaces can be developed by third parties to allow easy configuration of access control policies, which is both intuitive and easy to understand for the users; the usability aspect of such interfaces is beyond the scope of this work.
2. For each data entity – both structured and unstructured – input by a data owner to an xAccess-compatible web application, the data owner has the option to attach a category. Such categories are pre-defined based on the access control model specified in Step 1. For the default base model, these categories correspond to the set of defined roles. If any other access control model is used, any category attached to a subject or an object is transformed to its corresponding role in the base xAccess model. For example, the CBAC model allows user policies like "My parents should not see my party pictures". The CBAC system has the capability to determine if a particular picture is a party picture based on the its pixel contents [14]. A typical CBAC interface provided to a picture's owner would allow him to upload the picture to the web application. The CBAC system accordingly tags the picture by passing a role mapped to the "party picture" category to the xAccess extension. Finally, the mapping of the entities with their corresponding roles is passed to the application's server to be stored by the server-side component of xAccess.
3. Any user of a particular web application can view the default public information (i.e., public profile, public blog, etc.) of all other users. However, in order to access non-public information, the seeker sends a request for access to the owner of that information. This logic is internal to the application and is similar in most typical applications.
4. A data owner sees all the access requests from other users, and associates categories to each requesting user according to the access control model and the categories chosen by him in Step 1. The assignment of roles to the requesting users is conveyed to the server-side of xAccess. Again, this procedure is aligned with most existing applications where a user needs approval from a profile's owner before accessing the profile.
5. The requesting seeker can now access the data of another user (the data owner) in accordance with the roles assigned to him. The access control enforcement is done by the xAccess component on the server side using the algorithm given in Figure 4.
6. A data owner can modify the category of any of the objects contributed by him and any of the subjects approved by him, effectively changing the access allowed to these subjects.

4. EVALUATION

4.1 Prototype System and Applications

We developed a working prototype of the xAccess system, which includes the server-side platform code and APIs for integrating the access control framework into the web applications. We also implemented the labeling model that provides the required abstraction at the server so that any

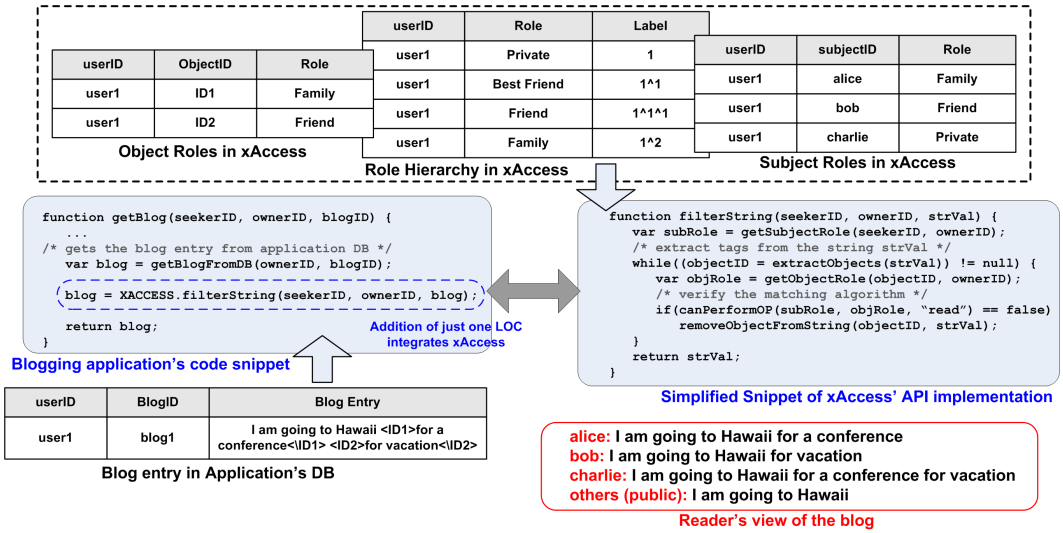

Figure 5: Blog example with sample xAccess API implementation.

access control model implemented at the browser side for the data owner can be enforced. Our xAccess platform consists of about 2500 lines of Javascript code.

We demonstrate the viability of our approach by means of two sample applications. First, we developed a blogging application in-house that stores the profile information for users and provide them the functionality to write their personal blogs. Second, we use a popular open-source wikipedia application, called mediawiki, to show the applicability of our system directly to existing applications. By default, these applications have open access with any user logged into the application being able to view the profile and data (blogs/wikis) contributed by other users. In order to show the feasibility of our approach, we integrated the xAccess platform into the server side of these applications. This integration represents one-time installation of our xAccess framework for any application supporting the generic access control models. We also showed that such an integration incurs minimal changes to the existing code of the applications; the change comprises of few xAccess API calls to filter the data being passed to the user.

On the browser side, we developed a sample proof-of-concept access control model that allows a user to design his own customized hierarchy of access control. The sample model is developed as a browser extension and has been tested for Firefox 3.5. Using the extension, users can perform addition, deletion or modification of new categories of access and customize the hierarchy graph of these defined categories. Our xAccess framework maps these user-defined categories to the abstractions at the server side using the APIs given in Appendix A. The example blogging application and the wikipedia application integrated with our xAccess framework can be accessed online at [3] and [1] respectively.

To handle unstructured data such as blogs and wikis, our extension allows users to attach a category by selecting text on their browser window. This enables the user to select complete or part of the blog or the wiki. A new identifier tag is attached to the selected text and the corresponding identifier to category mapping is stored.

Figure 5 shows a partial overview of our server-side implementation by means of our blogging application example. In this example, a user *user1* has three subjects, namely *alice*, *bob* and *charlie*, which the user has assigned categories of `Friend`, `Family` and `Private`, respectively. He has defined the role hierarchy for his system as given in Figure 3. xAccess stores numbered representation for the role hierarchy in its database. Taking an example, role `Private` (represented as 1) has two children `Best Friend` and `Family`, labeled as $1 \wedge 1$ and $1 \wedge 2$, respectively. By separately storing the role hierarchy, we allow modification of the hierarchy without requiring any changes to the roles already assigned to the subjects and the objects.

Let us assume that the user makes a blog entry saying "I am going to Hawaii for a conference for vacation" and wants to provide different purpose of his travel to friends and family. The user tags parts of the blog with different categories to achieve his purpose using the default interface provided by the xAccess extension. The modified text is stored in the application's database and the corresponding categories in xAccess objects' database table. By tagging the text with identifiers instead of the actual roles, xAccess facilitates access modifications on the text objects without changing the actual text stored with the application.

For any user accessing this blog, the server side of the application invokes the `filterString()` API of the xAccess platform (Appendix A) before passing the returned value to the requesting user. Note that this requires addition of only one line to the application code. `filterString()` filters the blog entry by invoking xAccess' access control algorithm according to the roles assigned to the individual text objects and the requesting subject. As we can see from Figure 5, different readers have their own views of the blog based on their assigned roles. While *alice* sees the entry as "I am going to Hawaii for a conference", *charlie* can view the whole blog entry as his `Private` role is higher in hierarchy to both tags `Friend` and `Family` attached to different parts of the entry.

We use similar text tagging method to control access for structured data that are text fields, even though the interface provided to the user is different. For non-text fields

Application Component	Operation	Number of access checks	Application Latency	
			Mean	Std Dev
User Profile (structured)	Read	12	37.1ms	0.83ms
Blog Entries (unstructured)	Read	5	16.9ms	0.60ms
Wiki Entries (unstructured)	Read	5	1.8ms	0.18ms
Wiki Entries (unstructured)	Write	5	5.8ms	0.46ms

Table 1: User latency of various operations in typical web applications with xAccess.

such as photos, the categories are assigned to the filenames. The web application uses the `isAccessAllowed()` API to verify if access is allowed, before passing these entities to the requesting seeker.

4.2 Performance Estimates

xAccess does not impose a substantial burden on the performance of the web applications. Without being able to deploy the framework to a real-world application setup, it is difficult to accurately predict the impact of our design on the performance of these applications as perceived by the users. To get a rough estimate of the cost of supporting the xAccess design and the overhead involved in our system, we conducted some experiments with our sample applications, measuring the latency introduced by added security provided by xAccess' access control mechanisms.

In our experiments, the xAccess server is hosted on a 2.4GHz Intel Quad Core 2 machine with 4GB of RAM. The requests are made from Firefox 3.5 browser on a 2.33GHz, 2GB RAM, Pentium Core Duo laptop. Each test was run 10 times and measurement values were averaged. We define user latency as the difference in the time when the request is made at the browser and the time at which the response is received by the browser. Table 1 shows the latency introduced for various user interactions for the sample applications. Each interaction performs a different amount of processing based on the number of access control checks made for the interaction. For example, the number of checks required for the user profile is fixed at 12, one each for every profile field. On the other hand, the filtering of the blog (or the wiki) depends on the number of different access control tags added by the user in the unstructured blog (or wiki) entry. In our tests, each of the sample blog and the wiki entry had 5 such access checks. Our results show that the user latency for applications employing xAccess for providing access control is still considerably low: when averaged over the number of access checks, the latency is about 3.1ms for structured user profile fields and 3.4ms for the unstructured blog. For the open-source wikipedia application, providing access control for the unstructured wiki incurs an average latency value of 0.4ms for read and 1.2ms for write operation.

Studies have shown that acceptable user latencies fall in the range of 50–150 ms [26]. All user latencies observed in our experiments currently fall within this range. However, the latency increases with the number of checks added by the user. We emphasize that our prototype implementation is written in Javascript with no emphasis on optimization. There are many opportunities for improving the performance in our system, by optimizing the database queries or by utilizing server-side caching. Moreover, on a cluster of commercial servers with much better computational capacity, these values will be even smaller. Although it is not possible to precisely determine the cost of our approach without a large scale experiment, both the details of our design and the results from these experiments, support the conclusion that xAccess design imposes additional latency within acceptable limits.

5. DISCUSSION

The support for both structured data (e.g., user's profile) and unstructured data (e.g., blogs) improves the ability of our framework to be acceptable in more diverse applications and environments. As previous research has suggested [13], ability to apply user-defined policies to a more finer grained level, such as words or phrases in blogs, satisfies a key access control requirement in the Web 2.0 paradigm. Other potential applications of our framework include email communication and newsgroups where a sender is passing messages to multiple receivers, either to anonymous groups or to unmanageably huge lists. Our framework allows the sender to attach desired access control tags to different parts of a message without explicitly identifying each receiver and creating separate individual messages. The filtering in turn is done by the sender's email server.

It might be argued that our design of pushing the access control models to the user's client might limit the mobility of the data owner, i.e., the ability to retrieve his access control preferences from the browser of any machine. However, the solution to this limitation could be trivial: since the web application is already storing the roles and role hierarchy for the user, it can allow downloading of such preferences to the user's client after login. Moreover, this is only required if the user wants to use a new machine to modify his access control preferences. In case no such modifications are desired by the user, he can normally browse the web application with no need for xAccess' browser extension.

While our abstraction model of access control can simulate a wide range of access control models and, in consequence, the variations of these models [6, 17, 18, 29], there is still a possibility that our design might restrict some other models that cannot be mapped directly to the abstractions presented by our framework. Since RBAC is policy neutral and research has shown its ability to successfully simulate various other models [15, 21, 23], and since our model is a variation of RBAC, we believe that our model also has the ability to cover many other popular access control models. We plan to evaluate the compatibility of our base model abstractions with other access control models as part of our future work.

6. RELATED WORK

Access control is a well established area of research and

previous work in this field range from creation of new models [6,10,24] to improving improving various aspects of these models [17–19, 29]. In this work, our focus is not on creating yet another all-purpose access control model but is on developing a generalized framework to allow such models to be integrated into web applications.

Access control for web applications and services is an area that has been well studied in the literature [7,9,20]. While such centralized solutions are well suited to the web applications, their design is limited in the changing Web 2.0 paradigm towards user-contributed data. Our work is more user centric allowing users to contribute in defining the access control for their own data.

The need for user-defined policies for user applications is being understood [13, 27]. PinUP is one system that allows users to manage file access of his own applications in an isolated environment specific to the user [11]. In another work, Simpson has argued that the users of social networking sites should have the opportunity to construct fine-grained access control policies that meet their particular requirements and circumstances [27]. Chinaei et. al proposed a decentralized access control system in which corporate policy can allow all health record owners to administer access control over their own objects [8]. Our work is not specific to any type of application and can be integrated into a wide range of web applications, including social networks, content sharing sites, and many others.

Gates has proposed use of access control based on the real-world relationships of users [13]. In agreement with our work, she also favors access control decisions to be made by the users regarding access to their data. She further argues the access control policies and relationship groups defined by the user should follow the user, rather than be redeveloped for each individual site. While her ideas relates to some of the work described in this paper, there is no real implementation of the ideas presented in her work.

There are some proposed solutions that allow inter operability between diverse web applications by using user-centric identities for access control mechanism. Lockr is an access control system based on social relationships that lets people manage their social networks by themselves in one place (e.g., through their personal address books) while letting web sites and Internet systems be in charge of content delivery only [28]. This eliminates the need for users to maintain many site-specific copies of their social networks. Similar protection is achieved by the single sign-on decentralized model of OpenID [2].

7. CONCLUSIONS

We presented a generic access control framework, called xAccess, that allows users to control how they want their data to be accessed. On one hand, xAccess is generic enough to enable users to choose their own access categories and on the other hand, it also supports integration of other access control models to further increase the diversity of policies available to the users. Our framework allows users to utilize a single unified access control across multiple web applications. From an application's prospective, it enables the application to support different access control models deployed by its users using a single model abstraction.

We developed a working prototype of the system and showed its viability by integrating two sample applications with our framework using the xAccess APIs. The blogging application was developed in-house and represents a typical web application with need for providing access control to its users [3]. We also integrated our framework into a popular open-source wikipedia application [1].

Our system shows promise in supporting potentially valuable future access control models that could be targeted by individual users to control access to their data. Since no change is required to deploy a new model after the one-time installation of our framework, any future models chosen by the users can be easily integrated to any web application.

8. REFERENCES

[1] mediaWiki with xAccess. `http://www.own-provide-decide.com:8001/mediawiki`.

[2] OpenID. `http://openid.net`.

[3] xBlog with xAccess. `http://www.own-provide-decide.com/accesscontrol/static/CLIENT/login.html`.

[4] M. Benantar. *Access Control Systems: Security, Identity Management and Trust Models*. Springer-Verlag, Secaucus, NJ, 2005.

[5] E. Bertino, J. Fan, E. Ferrari, M.-S. Hacid, A. K. Elmagarmid, and X. Zhu. A hierarchical access control model for video database systems. *ACM Transactions on Information Systems*, 21(2):155–191, 2003.

[6] J.-W. Byun, E. Bertino, and N. Li. Purpose based access control of complex data for privacy protection. In *SACMAT*, Stockholm, Sweden, June 2005.

[7] B. Carminati, E. Ferrari, and A. Perego. Enforcing access control in web-based social networks. *ACM Transactions on Information and Systems Security*, 2008.

[8] A. H. Chinaei and F. W. Tompa. User-managed access control for health care systems. In *Secure Data Management Workshop*, Trondheim, Norway, Sept. 2005.

[9] E. Damiani, S. D. C. di Vimercati, S. Paraboschi, and P. Samarati. A fine-grained access control system for XML documents. *ACM Transactions on Information Systems Security*, 5(2):169–202, 2002.

[10] D. E. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5):236–243, 1976.

[11] W. Enck, S. Rueda, J. Schiffman, Y. Sreenivasan, L. S. Clair, T. Jaeger, and P. McDaniel. Protecting users from "themselves". In *ACM Workshop on Computer Security Architecture*, Fairfax, VA, Nov. 2007.

[12] D. F. Ferraiolo, J. A. Cuigini, and D. R. Kuhn. Role-based access control (RBAC): Features and motivations. In *ACSAC*, New Orleans, LA, dec 1995.

[13] C. Gates. Access control requirements for web 2.0 security and privacy. In *W2SP Workshop*, Oakland, CA, May 2007.

[14] M. Hart, R. Johnson, and A. Stent. More content – less control: Access control in the web 2.0. In *W2SP Workshop*, Oakland, CA, May 2008.

[15] Y.-Z. He, Z. Han, and Y. Du. Configuring RBAC to simulate bell model. In *International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, Harbin, China, Aug. 2008.

[16] A. Lenhart and S. Fox. Bloggers: A potrait of the internet's new storytellers, July 2006. `http:`

//www.pewinternet.org/~/media//Files/Reports/2006/PIPBloggersReportJuly192006.pdf.pdf.

[17] N. Li and Z. Mao. Administration in role-based access control. In *ASIACCS*, Singapore, Mar. 2007.

[18] N. Li and M. V. Tripunitara. On safety in discretionary access control. In *IEEE Symposium on Security and Privacy*, Oakland, CA, May 2005.

[19] N. Li, M. V. Tripunitara, and Q. Wang. Resiliency policies in access control. In *CCS*, Alexandria, VA, Oct. 2006.

[20] M. Mecella, M. Ouzzani, F. Paci, and E. Bertino. Access control enforcement for conversation-based web services. In *WWW*, Edinburgh, Scotland, May 2006.

[21] S. Osborn, R. Sandhu, and Q. Munawer. Configuring role-based access control to enforce mandatory and discretionary access control policies. *ACM Transactions on Information and Systems Security*, 3(2):85–106, 2000.

[22] P. Samarati, E. Bertino, and S. Jajodia. An authorization model for a distributed hypertext system. *IEEE Transactions on Knowledge and Data Engineering*, 8(4):555–562, 1996.

[23] R. Sandhu and Q. Munawer. How to do discretionary access control using roles. In *ACM Workshop on Role-Based Access Control*, Fairfax, VA, Oct. 1998.

[24] R. S. Sandhu. Lattice-based access control models. *IEEE Computer*, 26(11):9–19, 1993.

[25] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[26] B. Shneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Addison-Wesley, 3^{rd} edition, 1998.

[27] A. Simpson. On the need for user-defined fine-grained access control policies for social networking applications. In *Workshop on Security in Opportunistic and SOCial networks*, Istanbul, Turkey, Sept. 2008.

[28] A. Tootoonchian, K. K. Gollu, S. Saroiu, Y. Ganjali, and A. Wolman. Lockr: Social access control for web 2.0. In *Workshop on Online Social Networks*, Seattle, WA, Aug. 2008.

[29] Z. Zhang, X. Zhang, and R. Sandhu. ROBAC: Scalable role and organization based access control models. In *CollaborateCom*, Atlanta, GA, Nov. 2006.

APPENDIX

A. SET OF XACCESS APIS

API	Description
`filterString(seekerID, ownerID, string)`	filters the *string* text according to the access control model of *ownerID* to give assess to *seekerID*
`isAccessAllowed(seekerID, ownerID, op)`	verifies if *seekerID* is allowed to perform the operation *op* on *ownerID*'s data
`setSubjectRole(ownerID, userID, role)`	assigns the *role* for *userID* in *ownerID*'s list
`setObjectRole(ownerID, objectID, role)`	assigns the *role* for *objectID* in *ownerID*'s list
`setRoleHierarchy(ownerID, roleHierarchy)`	updates the role hierarchy for *ownerID* on the web server

Table 2: Set of xAccess APIs for integrating the generic access control model.